Scope Graphs: The Story so Far

Aron Zwaan Hendrik van Antwerpen

April 5, 2023

Delft University of Technology

Names in Programs

- Programmers Love Abstractions!
- Perhaps most popular: Names
  - Abstract over Memory Locations
  - Later: Modules, Types, etc.
- Many Tools Manipulate Names

Audience Poll

This Java program is ...

A. well-typed.

B. well-typed if any class A exists.

C. untypeable, there is not enough information.

D. ill-typed.

```
class B extends A {
  B self() { return this; }
}
```

Audience Poll

This Java program is ...

A. well-typed.

B. well-typed if any class A exists.

C. untypeable, there is not enough information.

D. ill-typed.

```
class B extends A {
  B self() { return this; }
}

class A {
  class B { }
}
```

Audience Poll

This Java program is …

A. well-typed.

B. well-typed if any class A exists.

C. untypeable, there is not enough information.

D. ill-typed.

```java
class B extends A {
  B self() { return this; }
}

class A {
  class B { }
}
```

Audience Poll

This Java program is ...

A. well-typed.

B. well-typed if any class A exists.

C. untypeable, there is not enough information.

D. ill-typed.

```java
class B extends A {
  B self() { return this; }
}

class A {
  class B { }
}
```

```
$ javac B.java
B.java:2: error: incompatible types: B cannot be converted to A.B
  B self() { return this; }
                    ^
```

Traditional Approach: Environments

$$\dfrac{\dfrac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{T-Var}}{\begin{array}{c}\vdots \\ \Gamma; x : \tau \vdash e : \tau\end{array}}$$

$$\dfrac{\Gamma; x : \tau \vdash e : \tau}{\Gamma \vdash \lambda x : \tau . e : \tau \to \tau'}\ \textsc{T-Lam}$$

Traditional Approach: Environments

$$\dfrac{x:\tau \in \Gamma}{\Gamma \vdash x : \tau}\ \textsc{T-Var}$$

$$\vdots$$

$$\dfrac{\Gamma; x:\tau \vdash e : \tau}{\Gamma \vdash \lambda x:\tau.e : \tau \to \tau'}\ \textsc{T-Lam}$$

Binding Propagation follows Derivation Tree.

Scaling to Real-World Languages

- Non-Lexical Binding
- Possible Encodings
  1. Low-Level: Explicit Staging
  2. High-Level: Non-Algorithmic

```
class A {
  static void f() { B.g(); }
}
class B {
  static void g() { A.f(); }
}
```

Scaling to Real-World Languages

- Non-Lexical Binding
- Possible Encodings
  1. Low-Level: Explicit Staging
  2. High-Level: Non-Algorithmic

```
class A {
  static void f() { B.g(); }
}
class B {
  static void g() { A.f(); }
}
```

Cannot have Declarativity and Executability!

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A_1 {
  def x_2 = 1
}
module B_3 {
  import A_4
  def y_5 = x_6
}
```

Solution: Scope Graphs

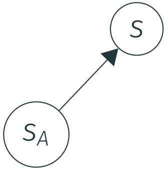
- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

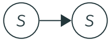
```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```



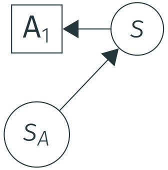


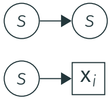
Lexical Parent

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```

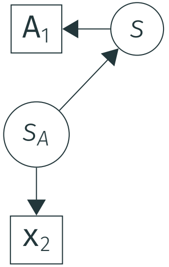


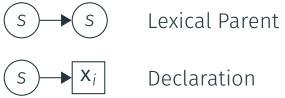


Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```

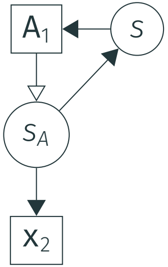


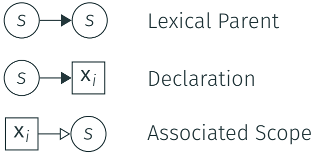


Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```

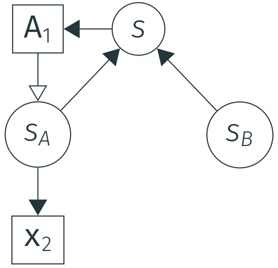


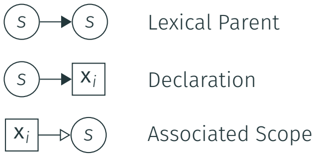


Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A1 {
  def x2 = 1
}
module B3 {
  import A4
  def y5 = x6
}
```

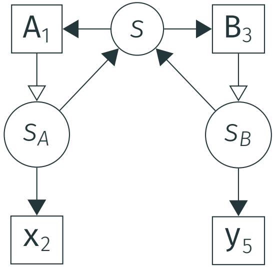


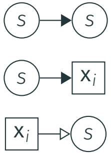


Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```





Lexical Parent

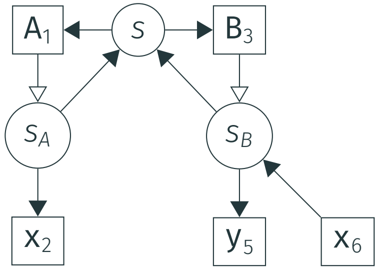
Declaration

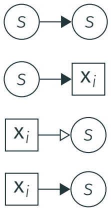
Associated Scope

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```





Lexical Parent

Declaration

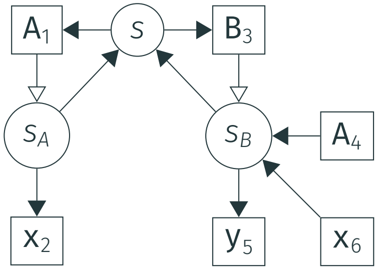
Associated Scope

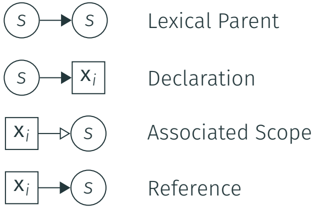
Reference

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A1 {
  def x2 = 1
}
module B3 {
  import A4
  def y5 = x6
}
```

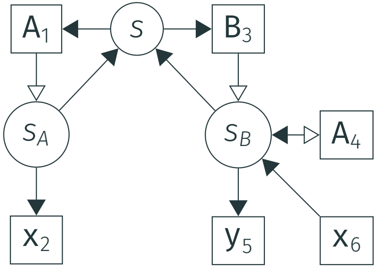


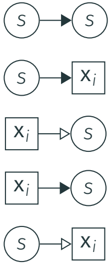


Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```





Lexical Parent

Declaration

Associated Scope

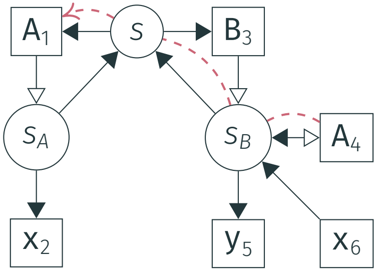
Reference

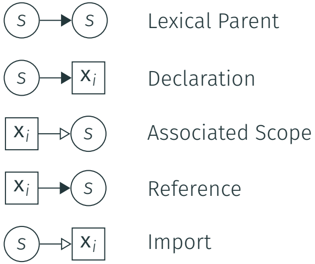
Import

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```

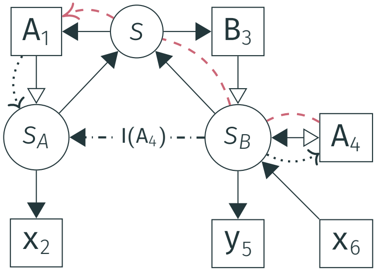


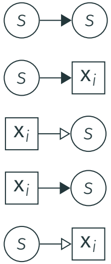


Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```



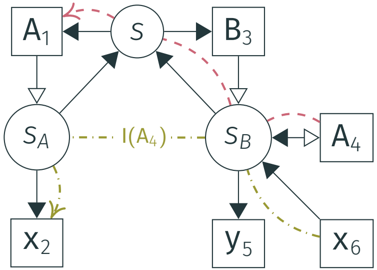


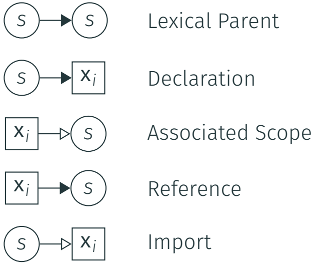
$s \rightarrow s$	Lexical Parent
$s \rightarrow x_i$	Declaration
$x_i \rightarrow s$	Associated Scope
$x_i \rightarrow s$	Reference
$s \rightarrow x_i$	Import

Solution: Scope Graphs

- Lexical: Tree-shaped Name Distribution.
- Non-Lexical: Generalize to Graphs!

```
module A₁ {
  def x₂ = 1
}
module B₃ {
  import A₄
  def y₅ = x₆
}
```





What Problems do Scope Graphs Solve?

- Declarative Specification of Name Binding
- Generating Type Checkers (Scheduling)
- Support Reasoning about Type-Safety
- Support Tooling: Editor Services/Refactorings

Overview: See Paper

Scope Graphs	Type Checking	Editor Services	Dynamic Semantics
Theory of Name Resolution Néron et al., 2015 [10]			
Constraint Language for Static Semantics van Antwerpen et al., 2016 [17]			Scopes Describe Frames Bach Poulsen et al., 2016 [14]
Scopes as Types van Antwerpen et al., 2018 [18]			Intrinsically-Typed Interpreters Bach Poulsen et al., 2018 [15]
Knowing When to Ask Rouvoet et al., 2020 [16]		Towards Semantic Editor Services Pelsmaeker et al., 2019 [12]	Meta-Interpreter Specialization Vergu et al., 2019 [21]
	Scope States van Antwerpen and Visser, 2021 [19]		
Specializing Queries Zwaan, 2022 [27]	Incremental Type-Checking Zwaan et al., 2022 [28]	Static Semantic Code Completion Pelsmaeker et al., 2022 [11]	

Figure 1 Overview of publications related to scope graphs.

Conclusion

Names are Ubiquitous in Programs.

Name Binding is Intricate.

Scope Graphs Support Versatile Program Analysis/Transformation Tools.